# XWiki Concerto: A P2P Wiki System Supporting Disconnected Work

Gérôme Canals[1] and Pascal Molli[1] and Julien Maire[1] and Stéphane Laurière[2] and Esther Pacitti[3]and Mounir Tlili[3]

[1] Nancy-Université, INRIA, France
{*firstname.surname*}@loria.fr
[2] XWiki, France
slauriere@xwiki.com
[3] Université de Nantes, LINA and INRIA, France
{*firstname.surname*}@univ-nantes.fr

**Abstract.** This paper presents the XWiki Concerto system, the P2P version of the XWiki server. This system is based on replicating wiki pages on a network of wiki servers. The approach, based on the Woot algorithm, has been designed to be scalable, to support the dynamic aspect of P2P networks and network partitions. These characteristics make our system capable of supporting disconnected edition and sub-groups, making it very flexible and usable.

**Keywords:** Collaborative Editing, Web based cooperation tool, Wiki, Distributed System, Eventual Data consistency

## 1 Introduction

Wikis are actually the most popular form of collaborative editors. Current wikis are built over the traditional centralized architecture of the web: the whole set of pages of a wiki reside on a single server. Some recent researches have proposed to shift to fully decentralized wikis relying on peer-to-peer networks [1, 2]. Expected benefits of this new approach are scalability, better performance, fault-tolerance, infrastructure cost sharing, self-organization, better support to nomad users and resistance to censorship [3].

The goal of the XWiki Concerto project is to design and implement the P2P version of XWiki[4], an open-source wiki system. Our vision of a P2P Wiki is a distributed system composed of a P2P network of autonomous wiki servers. The set of wiki pages is replicated over the whole set of servers. A change performed on a wiki server is applied to the local copy and then propagated to the other sites. Upon reception, a remote change is merged with local changes and then applied to the local copy. As in any P2P network, membership is dynamic and a wiki server can join or leave the network at any time.

We envisage 3 main use cases for our system: (1) Massive collaboration: wiki pages can be edited by very large groups of participants ; (2)disconnected work without restriction: any server disconnected from the network should be able to continue offering the whole wiki service, including wiki page edition ; (3) opportunistic collaboration:

[4] http://www.xwiki.com

a set of disconnected users should be able to create on-the-fly ad-hoc networks by connecting their wiki servers. They can thus collaborate and share their modifications inside this isolated subgroup.

There is two main issues in designing the P2P version of XWiki. The first is to ensure consistency of replicated wiki pages. The main challenge here is to design a replication mechanism that supports our three use cases and that offers guarantees on the consistency of the pages. The second issue is related to legacy: the XWiki server is an already existing software and the introduction of replication capabilities should have no impact on existing code (servers) and data (wikis).

The paper is organized as follows: next section goes in deeper details about requirements and discusses the state of the art about replication technics. Section 3 presents our approach to wiki page replication. Section 4 introduces the architecture of the replication component and of the XWiki Concerto system and decribes how disconnected work is supported in our system. Finally, section 5 concludes and discusses some future work.

## 2 Requirements and State of the Art

As presented in section 1 the first issue in implementing our P2P wiki is to ensure consistency of the replicated wiki pages. In other words, it is necessary to introduce a correctness criterion and a replication protocol that: (a) is scalable, to support massive replication in large dynamic groups of wiki servers, (b) supports asynchronous updates, including updates issued on disconnected servers, (c) supports network partitions to allow the creation of isolated sub-groups, (d) fits the requirements of wiki page edition: a wiki is basically an online editor for text content, and (e) can be used to replicate existing wikis with a minimum impact on the existing XWiki server.

Traditional pessimistic replication technics implemented in many distributed database servers are based on a correctness criterion commonly known as *one-copy serializability* [4]. However, this criterion and the existing protocols (e.g. ROWA, Quorum) do not fit our requirements, in particular scalability and the support of disconnected sites.

Optimistic replication technics rely on a relaxed correctness criterion called *eventual consistency* [5]: the system is consistent if all copies eventually converge to a common state when the system is iddle. Applied to our wiki context, we consider that a P2P wiki system is correct if it ensures eventual consistency and users intentions are preserved [6]. This last point takes into account the fact that a wiki is an editor: the convergence state can not be arbitrary but should keep all modifications introduced by all users and preserve their intentions. It is worth noting that copies can be temporarily divergent.

All optimistic replication approaches are based on the following scheme: (1) local operations (or group of operations) are immediately applied to the local replica, (2) they are thus disseminated to to the other sites, (3) upon reception, a remote operation (or a group of remote operations) are merged to the local history to take into account local modifications or other remote modifications - this may require the transformation of the remote operation, and (4) the resulting operation is applied to the local replica.

Following this scheme, implementing a P2P wiki can be done by adding to each wiki server an optimistic replication module made of two main components: a dissemination component, in charge of propagating group of operations (i.e. wiki *patches*) to the network, and a merge component, in charge of integrating remote patches to the local history. Of course, there is some constraints on these components:

1. the dissemination component must offer guarantees about the delivery of all patches to all servers in the network, including temporarily disconnected servers.
2. the merge algorithm need to be deterministic. Ideed, merges will occur on each site of the network and should give the same result on each site to guarantee the replica convergence. In particular, this avoid any user-based merge approach.
3. the merge algorithm need to be commutative and associative to support the delivery of patches in different orders at different sites.

There is different existing approaches for the merge. However, very few of them fit our requirements. For example, file synchronizers like Unison [7] are based on a non-deterministic merge because of the user involment in the process. In addition the approach requires the identification of a refence version. Merge are also widely used in Version Control Systems, either centralized like in CVS [8] or distributed, like Darcs or GIT. Centralized version systems guarantee the replica convergence, but are based on a reference copy handled by a central server that maintain a global order on versions. This approach has difficulties to scale up and cannot support disconnected subgroups. Distributed version control systems are more adequate to our concern. However, they have problems with consistency. Indeed, the *TP2 puzzle* scenarii [9] applied to this kind of tool lead to divergent replicas. Another interesting approach for merging is a family of algorithms based on Operational Transformation (OT) [10]. There exists a abundant contribution to this topic, for both real-time merge in group editors [11] or asynchronous merge in data synchronizers [12]. The strong point of these approaches is their genericity: they can handle different data types in the same framework. However, they generally make strong assumptions on the delivery ordering of disseminated operations. In most cases, these approaches use vector clocks to ensure a causal delivery, or a central scheduler to achieve a global order. These mechanisms clearly do not scale. Finally, a recent algorithm called Woot and specially designed for the P2P context has been introduced [13]. Woot will be presented in more details in the next section, since it is the algorithm on which we have built our system.

## 3 Page replication for XWiki Concerto

Our approach is based on extending the XWiki server with an optimistic replication mechanism that ensures eventual consistency. This mechanism is made of two main components: a dissemination component in charge of propagating patches to the network, and an integration component in charge of integrating patches to the local replica. We present in this section the design of these two components and their implementation in the XWoot application.

### 3.1 Patch dissemination

The dissemination component of the XWiki Concerto server is in charge of broadcasting patches to all servers in the overlay. Our component uses a classical P2P approach: it is based on a probabilistic gossip protocol called LPBCAST [14]. LPBCAST integrates dissemination and dynamic membership management in a single algorithm. Basically, each peer maintains a neighbour table. To broadcast a message, a peer first select a random subset of neighbours from this table and sends them the message. Upon reception, each neighbour does the same. The process is repeated a fixed number of

times. It is shown in [14] that by choosing correct values for the neighbour table size and the hop number, LPBCAST offers a very quick and very good probabilistic guarantee of delivery to all sites. In this algorithm, dynamic membership is achieved by piggybaging neighbour table content with messages. Upon reception of a message, a peer can then update its own neighbour table.

Our implementation of LPBCAST is very straightforward: patches are just packed with membership data into messages. Thanks to this dynamic membership management, a wiki server need just to know one single entry point (another wiki server) to join the replication network.

### 3.2 Patch integration: the Woot algorithm

The integration component of the XWiki Concerto server is in charge of integrating local and remote patches to the local copy. This component is based on the Woot integration/merge algorithm [13]. Woot is a merge algorithm designed for P2P settings. It is a specialized algorithm for linear structures like strings, so it is well suited for a wiki application that mainly manages page contents as text. Our choice is to manage concurrency at the line level, but it could be done at either the character or the word level. Woot is based on the following points:

- each line in the system receives a unique and non modifiable identifier at its creation time,
- the algorithm uses a specific storage model in which each line is represented as 3-uple: $(L_{id}, Visibility, Value)$. $L_{id}$ is the line identifier, and $Visibility$ is a boolean attribute that is True if the line is visible and False otherwise,
- there is only 2 operations : Insert and Delete. An update is thus expressed as a delete followed by an insert,
- the delete operation, $Del(L_{id})$ receives the identifier of the line to be deleted and just set its visibility attribute to False,
- the insert operation, $Ins(id_1 < Val < id_2)$ receives the value of the line to be inserted and the identifier of the line before and the line after the intended insertion position.

To integrate a set of operations to an existing Woot document, the algorithm builds a dependency graph between operations and lines. The algorithm then works by linearising this graph. Woot guarantees that the linearisation order is indepedent of the site on which it is computed and of the integration order of the operations. This ensures that Woot is a Deterministic, Commutative and Associative (DCA) merge algorithm. It requires no ordering constraints on the delivery of patches. This characteristic makes the approach very suitable for the P2P context: it is scalable (no use of vector clocks or centralized timestamp), completely decentralized (no global state or order) and can integrate patches at any time. It thus can support network partitions and disconnected operations.

## 4 Architecture and Support to disconnected work

### 4.1 Architecture

The architecture of our system has been designed with two objectives: (1) minimum impact on the existing code of XWiki and (2) to facilitate the disconnection from the

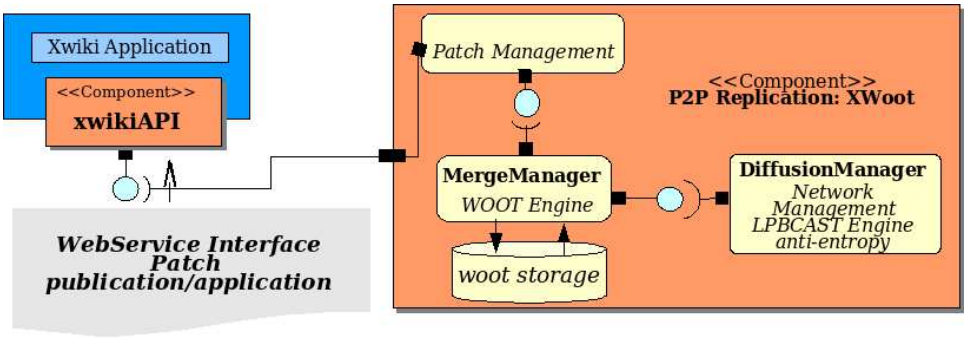


**Fig. 1.** XWiki Concerto architecture for replication

network. Our approach is based on a web SOA architecture where the replication component, called XWoot, is a separate application from the XWiki server. This application implements the Woot merge algorithm and its dedicated page storage. It communicates with remote XWoot applications by sending and receiving patches thanks to the dissemination component.

XWoot communicates with the XWiki server through a web API. The XWiki server is just extended to support this web API that allows it to publish pages locally modified and to apply new page values resulting from the integration of remote patches. This API implements 3 main methods:

- *getModifiedPages()*: returns the set of pages (identified by their Page Id) modified since the last call,
- *getPageContent(PageId)*: returns the current content of a page,
- *setPageContent(PageId, PageContent)*: sets a new page content for the page identified. This operation fails if a new version of the page has been produced by the XWiki server and not yet accessed.

This API is accessed by the XWoot application. The XWoot application periodically polls the XWiki server to check for new page versions. When a new page version is available, it is accessed by the XWoot application. A patch corresponding to the modification is then computed by XWoot by diffing it with the previous version. The resulting patch in then integrated to the woot storage and propagated to the network. When a remote patch is received, it is integrated to the woot storage. Then, the new page content is extracted from this storage and transferred to the XWIki server with the *setPageContent()* method. Of course, if a new version has been produced by the server since the last poll, this setPageContent fails and the XWoot application needs to get the very last version of the page. Once this version is integrated into the Woot storage, the new version resulting from the merge of the local version with the remote patch can be extracted and transferred to the wiki server.

The main interest of this architecture is that it has a minimum impact on the XWiki server. Indeed, the server is just extended with the page publication API. In addition, it allows to replicate existing wikis. When an XWoot application is connected for the first time to an existing wiki, the *getModifiedPages()* call will return all pages of the wiki. The XWoot application will then ask for the content of all pages, and store them in its own storage. These pages are then ready for replication.

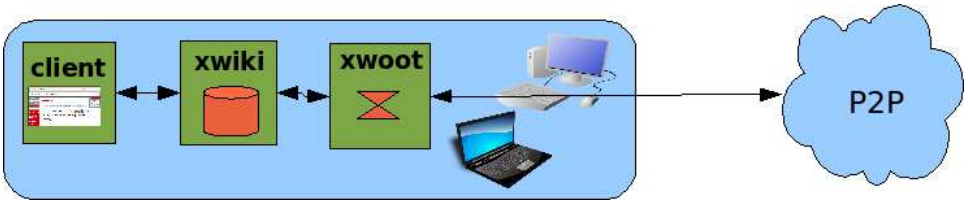


**Fig. 2.** XWiki Concerto Nomad deployment

### 4.2 Supporting Disconnected Work

The architecture introduced below is based on three components: the XWoot replication manager, the XWiki server, and the Client (usually a web browser). This architecture can be deployed in different ways:

1. The classical way: the pair XWiki/XWoot on a server, accessed by traditionnal web clients running on any king of device,
2. The nomad way: the three components run on the same device, generally a desktop or a laptop.
3. The mobile way: the XWoot replication manager runs on a desktop/laptop while a wiki page edition service is embedded on a mobile device.

**XWiki Concerto Nomad deployment** The nomad deployment (see figure 2) is of interest to support disconnected work on desktops/laptops. In this case, disconnected work is supported thanks to the local replica of the wiki pages. The disconnection of the XWoot application from the network is handled by an anti-entropy protocol [15]. This protocol allows an application that reconnect to the network to get patches it missed during a disconnected period. To do so, the XWoot application sends its local history of patches to one of its neighbours. This neighbour, upon reception of this log, computes the difference with its own log to determine the set of missing patches. It then sends back this set of patches to the anti-entropy initiator that just need to integrate these patches on its local replica. It is worth noting that this is also the protocol used to create a new replica on a node joining the network for the first time. This new node just starts the anti-entropy with an empty log and gets back the complete set of patches from its neighbour. This approach allows any user that wants to work offline to do so. He just need to start a new local XWiki/XWoot process and join the network to get a replica. He can then disconnect, continue to edit the wiki, and later reconnect at any point of the network. In addition, several disconnected users can create an ad-hoc overlay network disconnected from the main one and publish and exchange patches inside this sub-network. When they reconnect to the main network, each participant publishes the patches it produced and get the patches it missed during the disconnected period.

**XWiki Concerto Mobile deployment** The mobile deployment (see figure 3) is of interest to support disconnected work on a mobile device (pda, smart phone) : the XWoot application can rest on a desktop or a laptop, while a lightweight XWiki application is embedded on the mobile device. This specialized application offers page

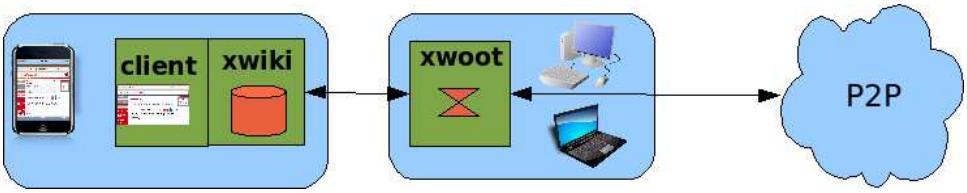


**Fig. 3.** XWiki Concerto Mobile deployment

edition and page rendering facilities. With this strategy, merges, which can be costly, are not run on the mobile device. However, it obviously requires that the XWiki application reconnects to the XWoot from which it disconnected. The disconnection between the embedded application and its XWoot companion is easy provided that the XWoot application remains connected to the network. While the mobile XWiki editor is disconnected, the XWoot application continues to receive remote patches and integrates them to its local store. When the XWiki editor is back, XWoot gets all modifications added during the disconnected period, merges them with the remote modifications, and then sets the new merged page contents.

## 5 Conclusion

We have presented the XWiki Concerto system, the P2P version of the XWiki server. This system is based on replicating wiki page on a network of wiki servers. The approach, based on the Woot algorithm, has been designed to be scalable, to support the dynamic aspect of P2P networks and network partitions. These characteristics make our system capable of supporting disconnected edition and sub-groups, making it very flexible and usable. A first experimental version of XWoot, the replication manager for XWiki has already been released. A product version is planned for the end of 2008. A lightweight version of the XWiki server is also being implemented and we are actually working on connecting this version with the replication manager. It will allow to edit wiki pages on a smart phone.

In a near future, we plan a version including security protocols, and in particular protocols for controlling access rights when joining an overlay network. This point however raises a lot a research questions and is a very open issue.

Two others open issues are actually under examination. The first concern is about performances. We are actually working on performance evaluations of the XWoot application, and in particular about the evaluation of the time required to achieve convergence, depending on the replica size, the number of patches, the network load ... We are conducting the evaluation by using the SNAP (http://snap.objectweb.org/) platform. The second concern is about partial replication. Actually, our approach is based on a complete replication of a wiki on all servers of the network. This is not desirable on all cases, particularly for privacy reasons. However, partial replication, i.e. replication of only parts of a wiki, may cause consistency mismatch when semantic dependencies (e.g. links) exists between replicated and non-replicated pages.

## 6 Acknowledgments

Authors want to acknowledge all the participants of the XWIki Concerto project for their fruitful contributions, and in particular Patrick Valduriez, from the INRIA ATLAS project, Ludovic Dubostand Sergiu Dimetriu from XWiki and Karim-Pierre Maalej from ENST.

## References

1. Morris, J.: DistriWiki:: a distributed peer-to-peer wiki network. Proceedings of the 2007 international symposium on Wikis (2007) 69–74
2. Weiss, S., Urso, P., Molli, P.: Wooki: a p2p wiki-based collaborative writing tool. In: Web Information Systems Engineering, Nancy, France, Springer (December 2007)
3. Androutsellis-Theotokis, S., Spinellis, D.: A survey of peer-to-peer content distribution technologies. ACM Comput. Surv. **36**(4) (2004) 335–371
4. Bernstein, P.A., Hadzilacos, V., Goodman, N.: Concurrency Control and Recovery in Database Systems. Addison-Wesley (1987)
5. Saito, Y., Shapiro, M.: Optimistic replication. ACM Computing Surveys (CSUR) **37**(1) (2005) 42–81
6. Sun, C., Jia, X., Zhang, Y., Yang, Y., Chen, D.: Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. ACM Transactions on Computer-Human Interaction (TOCHI) **5**(1) (1998) 63–108
7. Balasubramaniam, S., Pierce, B.: What is a file synchronizer ? In: Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking, MobiCom'98. (October 1998)
8. Berliner, B.: CVS II: Parallelizing software development. Proceedings of the USENIX Winter 1990 Technical Conference **341** (1990) 352
9. Li, D., Li, R.: Preserving operation effects relation in group editors. In: CSCW '04: Proceedings of the 2004 ACM Conference on Computer supported cooperative work, New York, NY, USA, ACM Press (2004) 457–466
10. Ellis, C., Gibbs, S.: Concurrency control in groupware systems. ACM SIGMOD Record **18**(2) (1989) 399–407
11. Sun, C., Jia, X., Zhang, Y., Yang, Y., Chen, D.: Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. ACM Transactions on Computer-Human Interaction (TOCHI) **5**(1) (1998) 63–108
12. Molli, P., Oster, G., Skaf-Molli, H., Imine, A.: Using the transformational approach to build a safe and generic data synchronizer. In: Proceedings of the ACM SIGGROUP Conference on Supporting Group Work - GROUP 2003, Sanibel Island, Florida, USA, ACM Press (November 2003) 212–220
13. Oster, G., Urso, P., Molli, P., Imine, A.: Data consistency for p2p collaborative editing. In: Proceedings of the 2006 ACM Conference on Computer Supported Cooperative Work, CSCW 2006, Banff, Alberta, Canada, November 4-8, 2006, ACM (2006)
14. Th., Guerraoui, R., Handurukande, S.B., Kouznetsov, P., Kermarrec, A.M.: Lightweight probabilistic broadcast. ACM Trans. Comput. Syst. **21**(4) (November 2003) 341–374
15. Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H.: Epidemic Algorithms for Replicated Database Maintenance. In: Proceedings of the 1987 ACM Symposium on Principles of Distributed Computing - PODC'B7, ACM Press (1987) 1–12